4

USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES
(AIRMICS)

DTIC FILE COPY

AD-A216 940

TECHNOLOGY ASSESSMENT OF COMPUTER LANGUAGES

(ASQBG-I-89-012)

February, 1989

DTIC
ELECTE
JAN 18 1990
S B D

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

90 01 17 021

This research was performed as an in-house project at the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). This effort was performed under the AIRMICS Technology Insertion Program to support the U.S. Army Information Systems Command (USAISC) in the development of a report entitled "Long Range Planning Guidance - Objective Configuration." An initial meeting was held in early December in Atlanta to coordinate the task. Twenty-six topics were selected for consideration, with AIRMICS agreeing to conduct technology assessments on fifteen of the topics. Planning Research Corporation (PRC) was assigned responsibility for conducting the remaining assessments and consolidating all the assessments for use in the planning document. In a two-week period, AIRMICS completed the assessments and provided the results to ISC-DCSPLANS and ISEC-SID. This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

INSPECTED COPY 1

s/ [illegible]
Glenn E. Racine
Chief
CISD

s/ [illegible]
John R. Mitchell
Director
AIRMICS

n For
&I ✓
☐
:ed ☐
ition

Distribution/
Availability Codes

Dist	Avail and/or Special
A-1	

Technology Assessment - Computer Languages

The purpose of this paper is to examine the history, assess the current trend, and then provide some insight into the future development of computer languages. To this end, four major sections are provided. Section I is a historical perspective of computer languages; Section II, the trend of today's development; Section III, a projection of what may be in the 1995 timeframe; and Section IV, the 2010 timeframe. Each section relies heavily on the information from the preceeding section, which is also typical of computer languages. Each subsequent language seems to build from the mistakes and strengths of its predecessors. [illegible]

I. A Historical Perspective

In its infancy, computing did not include the use of software. If one wished to add numbers, one pushed the mechanical "add" button after each entry, or wired the board in a certain manner. "Computing", known as Automatic Data Processing (ADP), was unforgiving of errors, regimented, and time consuming. But as the sophistication of hardware improved, software became necessary to aid in the control of data and operations. This necessity created an interest in software development and fostered efforts in that direction. This success was short lived. Numerous software improvements were being re-engineered into hardware or hardware-like characteristics (the precursor to the process known as firmware). Only with the advent of computer languages for the "user" was software able to achieve a distinct function within ADP. However, even with a distinct function, software did not play a primary role in shaping the computer industry of the future. In the late sixties as ADP was emerging as a true, needed technology, software had already assumed a secondary role to hardware. Definitions during that era of computer processing reflected this attitude. "... to the basic computer hardware is added the software of systems programs ... the hardware provides the ability to do basic operations, the software the ability to specify the job in a convenient notation ...". Software has yet to erase this stigma.

HOW TO REPORT HISTORY?

A presentation of historical information concerning computer languages can be conveniently divided into three parts, with a discussion of the three eras or phases when languages were at different degrees of use, stability and sophistication. The difficulties lie in labelling each stage and in agreeing upon its beginning and ending dates.

Using the term "generation" would appear to be a convenient word to identify the phases, but it conflicts with the definition of the levels of abstraction – the first generation language being machine code; the second generation language (2GL), assembler; 3GL, higher order languages; and so forth. Other terms are available but are equally plagued with opposing interpretation; therefore, labelling the phases will be limited to: first phase, second phase, and third phase.

The use of inception dates for higher order languages would be one method of delineating the time period for each phase. The first era covers the period 1950 to 1958 and includes the languages of FORTRAN I, Algol 58, Flowmatic, and IPL 5. The second era covers the period of 1959 to 1961 and includes the languages FORTRAN II, Algo 60, COBOL, and LISP. The third era covers the period of 1962 to 1969 and includes the languages PL/I, Algol 68, Snobol 4, Simula 67, Pascal, APL, and Basic. However, these timeframes do not reflect all the development periods of computer languages. For that reason and to be able to reveal various attitudes at the height of influence, historical phase dates for this paper will reflect the primary periods of use, 1947 to 1955, 1956 to 1965, 1966 to 1982. 1983 to the present will be discussed in the Current Trend information.

PHASE I

During the first phase, computer languages were typically computer dependent, having been designed by hardware developers. Most of the languages were crude and unwieldy because they ran on slow, single minded machines. If these early computers were to perform a different operation, the primary method employed was to stop the on-going task and initiate the new task. Machines achieving a higher quality status succeeded by manipulating the bits and bytes of the computer's registers through the use of machine code. Machine code was the language of the hardware. Machine code was also cumbersome and required an intimate knowledge of the internal architecture of the computer. What was needed was a "language" to incorporate several machine code instructions into one readable and understandable instruction, something other than punches on a papertape. That "something" became assembler languages, considered one step above machine code. However, the few advances obtained were not enduring. Most of the assemblers and all of the attempts at higher order languages of this era are not in use today. Two reasons come to mind. First, if the computer's architecture and/or operating system changed, usually because of a radical upscaling of the system, a new assembler language was needed. It was easier to completely rewrite the assembler than to modify the old. Second, those "assembler" languages that made it out of the lab and attempted to divorce themselves from a particular operating system or architecture (higher order language), did not have the needed support to survive. Most computers in public or private use were still research tools or experiments. Little interest was shown for one all-encompassing language. It was not until the second phase came into being that this attitude changed.

PHASE II

The second phase is characterized by major changes in hardware architecture and the founding of usable higher level languages. The major changes in hardware continued to cause havoc with computer language development, a noted reason for language failure during the first phase. However, because the focus of the computer developers changed, some stability and a sense of direction occurred in the computer industry. The new focus was the "user". Money could be gained by providing companies and the government a computer with which to do their mundane operations, but there was one major stumbling block in this economically motivated process. Users did not want to learn assembler. Thus was born the need for true higher order languages (HOL). HOLs such as FORTRAN and COBOL came into existence, went through enhancement revisions, and flourished in the scientific and business communities. Other languages were created but did not experience the success of FORTRAN and COBOL, although they certainly had their own important applications within the computer environment. To date, approximately one hundred and fifty languages and dialects exist that belong to this overall domain of general-purpose languages. Possibly as much as eighty-five per cent of the world's software is written in general-purpose languages.

PHASE III

The third phase could be characterized simply as newer versions of old efforts. Multiple facsimiles and dialects of FORTRAN, COBOL and other languages were in existence. National standards were needed; users wanted to share information, to access other programs. The result - a standardization effort came into being. Language experts attempted to provide more stable products. During this era, they were afforded an opportunity to consolidate their product instead of continually being forced to enhance it. Additionally, with the extensive modernization effort of computers came a similar improvement in the software industry. The newer computers boasted more memory and were much faster, allowing software to be less constrained as to the amount of memory it could utilize. Methodologies were also influenced.

The second phase programming technique of parcelling one's program into multiple subroutines gave way to keeping one large program and allowing the operating system to manage the size via paging. Software products and the manner in which they were built were becoming bigger and faster. No longer were hardware limitations completely dominating the artistry of language development and use. The stigma of software following hardware had its first glimmers of being reversed. The process of choosing hardware, selecting software, then determining requirements was being touted as wrong. Life cycle development processes were finally coming of age.

II. The Current Trend

PHASE IV

The current trend had its beginnings during the height of the third phase, approximately 1977. Due in part to general-purpose languages achieving most, if not all, of the functional accomplishments they were likely to achieve, computer languages branched into a new and exciting area. No longer would the programming environment be neatly divided into the two standard computer ideologies - low order (assembler) and high order languages. An industry within an industry was about to emerge. Microcomputers and their associated languages, operating systems, and specialized software were beginning to have their profound impact.

Not only did microcomputers revolutionize the computer hardware and software industry, they altered the method in which daily business was done. Prior to the micro era, the "ADP shop" consisted of language and hardware specialists. These ADPers controlled access to the main computer; therefore, they controlled the use and application of the business' information. The ability to perform assigned tasks at one's desk eliminated the need for an ADP shop to act as an intermediary between the user and the main computer. Enough computer power in terms of memory, storage and speed was now located at a user's desk when ten years earlier a smaller capacity, lesser capable machine would have filled the room the desk was in. Users were now "in charge"; the computer industry was bending to their demands. This fostered the growth of specialized software, tools, and experimentation with other forms of language.

AN EXPLANATION OF USER DEMANDS

A. SPECIALIZED SOFTWARE. Specialized software, or fourth generation language (4GL) software, comes in various and numerous forms. Best known are the spreadsheet and database packages for the personal computer (PC). Most would not be considered languages in the purest sense because they do not follow the format as defined for the first, second and third generation languages. The old generations considered a programming language to be instructions or commands that produce machine code, in a systematic, several step process. Usually the input had to be assembled and/or compiled, loaded, then executed. Some of the early language processors were even more simplistic, making use of an interpreter instead of a compiler. The interpreter executed statements in a language by decoding one statement and immediately executing it, then going to the next statement to repeat this process. 4GLs are not as meticulous. There is no single process each follows; logic can be built in different stages. Spreadsheets, for example, can update the information entered in each cell as it is placed in the cell or can wait until a function key is activated. Likewise, syntax errors do not cause the process to stop, but can be corrected by re-entering the data or command. This occurs because 4GLs are interactive. One enters a command or data and the 4GL package reacts to the entry. In essence, 4GLs on the PC combines database information with a specialized non-procedural language.

There are 4GL packages for larger configurations than PCs. The main thrust of these packages is to perform enhancement functions in the delivery of information. All can be self-sustaining functions. Examples are queries for databases, such as SQL (Structured Query Language); query languages for databases, such as FOCUS, RAMIS, or IDEAL; and HOLs for distributed systems programming, such as Monitor, Linda or NIL. Interestingly, most of these names were not derived by the builders of the product but by the marketing or sales staff. For example, SQL is neither structured, nor limited to queries, nor a language.

B. TOOLS. A new trend is occurring in the manner in which application software is built for some general-purpose HOLs. Less and less are software developers using the third generation languages, their associated utilities, and the traditional life cycle methodology to construct all software applications. The new approach of using 4GL tools is in use for small systems development, with the prospects of performing the same services for large systems once the maturity and support of 4GLs increase. Methodologies such as prototyping are flourishing under the 4GL influence. Prototyping is an iterative process of refining the design of an application by providing a quick "picture" of the product. But as quickly as one tool is mastered, another new improved tool appears to take its place. Application developers are being inundated by the volume and variety of tools. There appears to be no time to allow a product to mature, and as a result, there appears to be limited continuity of effort.

C. EXPERIMENTAL LANGUAGES. Those languages considered experimental have come to the forefront during the last few years. Most have been in existence for several years but are only now finding the support and an application for the language. The most interest has been shown in the following four classes of language:

1. Logic
2. Functional
3. Object-Oriented
4. Block Structure

A fifth class of languages (Special Purpose) will also be addressed. The potential impact of this class of languages has been, and will continue to be, of minor consequence; therefore, little discussion is provided.

1. LOGIC. Prolog (PROgramming in LOGic), about ten years old, has attracted the attention of a fairly large number of European computer scientists. Prolog is a declarative rather than a procedural language based on predicate calculus, with its foundation seated in Robinson's automatic theorem-proving theory. It is the language of choice for the Japanese in their fifth generation language efforts. PROLOG is considered a logic language because its statements are translated as logic phrases, not strictly text phrases.

2. FUNCTIONAL. Functional programming is the most difficult of the four programming techniques or languages to provide a discrete definition of its actions. The basic concept was proposed by John Backus. Functional programming (FP) consists of functional forms, and their functional style of programming facilitates the utilization of parallel machine architecture. FP has taken several twists and turns from its original understanding. Other "notions" and models exist, each with a different slant on what constitutes true FP. For example, LISP is considered as the first FP. LISP is a logic oriented, general-purpose programming language based on Church's lambda calculus. However, LISP does not have the full expressive power of lambda calculus and is thought by some language experts to be a logic-oriented language. Newer FPs are LOGO and SCHEME.

3. OBJECT-ORIENTED. Object-Oriented (O-O) languages have gained the most attention recently. Products such as C++, Smalltalk, and Objective C are impacting the development cycle for the low-end computer users. O-Os mirror some of the characteristics of tools, as complex applications can be developed easily using a minimal number of instructions. But features such as reusability, inheritance and typing distinguish O-Os as languages bordering on

the fifth generation class. As the usage, maturity and support of O-O rise, implementation on large systems may become viable.

4. BLOCK STRUCTURE. Block Structure or Data-Driven languages are extensions of general-purpose languages. The basic structure follows the lines of third generation languages; however, emphasis is placed on the interaction and tight control of data. The most noted language in this group is Ada. Ada has incorporated in its schema such functions as information hiding, strong data typing, and other software engineering properties. The Department of Defense (DOD) is the proprietor of the language, holding a registered trademark to the name "Ada" and, until recently, dictating compiler development and validation. The initial application of Ada was in the real-time, embedded systems environment, on-board computers of tanks, airplanes, ships, etc., but has been expanded to include new development in the Management Information Systems (MIS) or business functions area of the DOD. The debate continues as to Ada's usefulness. Contractors to DOD and DOD components have expressed concerns over the lack of integration or "bindings" to 4GLs and databases, and to the maturity of the Ada language compilers.

5. SPECIAL PURPOSE. As has occurred in other phases, languages have been created to satisfy particular functions. Most have been short-lived. Most fulfill a single purpose, with their accomplishments and usefulness usually being passed to another version to be improved. Pascal and Modula-2 are examples of this class of languages. During the seventies, Pascal was developed as an instructional language, with a strong emphasis on good programming style. Later, Modula-2 was built to improve on this premise, to include newer techniques. Pascal enjoys a strong following; Modula-2 is expected to do the same.

III. The Near Term (1995)

NO UNEXPECTED CHANGES

It will be "business as usual" in 1995 for languages. The information presented in the Current Trend section of this paper will, for the most part, still be valid. Improvements will be made in specific areas, those will be discussed in later paragraphs. The reason this projection can be made is the amount of effort and fiscal investment needed to change the methodical trodding of the current language trend can not be met. History has shown it takes eight to sixteen years, usually more towards the latter, for a computer language to acquire some dominance in the computer industry. A precedent was set with the microcomputer, establishing the lower boundary of the scale.

development process of the micro and its associated 4GLs took approximately six years and eight years respectively to influence the commercial marketplace. No hardware has been shown or has been predicted as having the magnitude of impact of the micro; and, since six years from the current date is the near term date, 1995, it is unlikely a major diversion will occur. This is not to say small corrections in the direction of language advancements will not be experienced. Those software disciplines most likely to show gains, already in the development time cycle, that may influence the daily business of a select few are:

1. Fifth generation languages
2. Expert system shells (tools environment)
3. Ada interfaces

1. FIFTH GENERATION LANGUAGES. As with several of the new technologies, a consensus definition of fifth generation languages (5GL) can not be obtained. Developers in the 5GL field tend to define the subject based on the tools being built. But the tools vary as widely as the range of services they perform. Most experts prefer to talk to the functions of 5GLs, all seemingly pointing back to a subset of the Japanese's 5GL project. When finally required to provide a definition, a simplistic definition is given, "knowledge processing". The first generation languages were considered number counters; the 2GLs were number crunchers; the third, information sharing processors; the fourth, workstation processing; and the fifth, knowledge crunching. 5GLs have been nicknamed, "silicon coworkers". In 1991, the several elements of 5GLs should be subsumable. During that year, the Japanese (the "leading edge" developers in this arena) are scheduled to complete their project.

2. EXPERT SYSTEM SHELLS. There will be a noticeable difference in the use and productivity between those who choose to stay with the current tools or their upgrades and those willing to try Expert System Shells (ESS). The ESS route is the preferred path. Much is to be gained; however, the complacent users will greatly outnumber the ESS users. Several surveys have been performed to determine if users would switch to improved products over what they currently use. When the worst case was presented, to be content to wait for vaporware (new product or upgrades to products promised but not delivered on-time) instead of switching, the majority said they would wait. From this disclosure, it would appear safe to assume the same mindset will be prevalent during the mid-nineties. That is, as a tool matures, the support base will stay with that tool instead of continually updating their inventory as newer tools and ESSs become available.

Additionally, the level of tool development at present is not as anticipated. The prediction was made that with the maturing of 4GL tools, the need for programmers would go away. Because sophistication has fallen short of the expectation, the prediction has now been updated to say, "with the coming of 5th or 6th generation language tools". As the applications for tools grow in function, ability, and purpose, they should evolve into ESSs. ESSs are self-contained entities. They are not like most tools - an intermediary - "front-ending" information into one of the third generation language HOLs. In the year 1995, tools will not have made the transition of being expert system shells, the demand not being what it should be. The influencing factor is the microcomputer. The small scale tools used on the micro - LOTUS 1-2-3, dBASE IV (or whatever the names of the derivatives) - will likely remain the mainstay of tool use.

3. ADA INTERFACES. Ada will have matured and have established a position within the military and parts of the commercial communities. The missing element, the lack of integration, will have been solved. This scenario is predicated on DOD expanding the use of Ada to more than the real-time environment. In fact, the real-time environment must become secondary in importance to the needs of the business environment if it is to succeed.

THE ROLE OF SPECIAL PURPOSE LANGUAGES

There will, in all likelihood, be new languages invented in the academic and research communities to solve functional or application specific problems. As with the earlier discussion of Pascal and Modula-2, the product's purpose may be as simplistic as teaching proper programming techniques for that era. It does not matter what the purpose is; what does matter is requirements will be driving the software, software will be driving the hardware. Users will become more adept at defining their requirements. If the languages or tools of the day do not satisfy their requirements, a new language or tool will be created.

WHERE ARE THE OLD STAND-BYS?

A discussion of the near term would not be complete without evaluating the status of COBOL, FORTRAN, and the other general-purpose languages. As expected, these keystones will still be in use. The usage level will not have grown. Development of new applications will have diminished, or possibly, to have completely stopped. With all the new tools, shells, etc., it would not appear to be productive to generate 3GL-based products. Only mild support of these languages can be anticipated, with vendors offering a trade-in incentive to migrate to a 4GL or later product. The function will remain but the market will have disappeared.

IV. The Long Term (2010)

THE Nth GENERATION LANGUAGE

What generation of language will be the current version in the year 2010? In the 1995 discussion of tools, an allusion was made to the sixth generation. No discussion nor definition was provided. None could be found in the literature searches performed that had any substance. If a substantive definition had been found, it is uncertain if it would have been understandable. It was difficult enough pinpointing and understanding a definition of fifth generation languages. To this extent, the discussion of the Nth generation would have little meaning other than to say the industry is taking the next step beyond the knowledge processing class of generation languages. To date, computers have required total information before an intelligent response could be provided ("garbage in, garbage out"). Is the 6th or Nth GL the basis for a thinking machine? One that does not need total information, complete information, or prior knowledge before a decision can be made. Instead of a silicon coworker, you might be working for a silicon boss.

Those in the AI environment believe all languages and tools will evolve into AI disciplines by 2010; it is unlikely that this will be the case. COBOL and FORTRAN, whatever their form, will still be in use. The trend of 1995, replacing the general-purpose language applications as they lose their usefulness, will be on the rise. But the numbers are just too great. The possibility remains these general-purpose languages will be migratable to another third generation language - Ada. If the support and services are directed towards Ada and the incentives are there, the demise of COBOL and FORTRAN may come sooner. Ada is expected to undergo revisions; however, the process is typically slow and tedious.

THE OTHER PLAYERS

Some techniques will have matured and established a niche within the computer environment. Their part will be small but needed; for example, distributed processing. The basis for distributed processing is establishing an operating system to direct the processing and handling of information over several computers. A process akin to parallel processing. Information in this environment can be located on several computers at multiple sites, each with a distinct and separate portion of the "knowledge base". Duplication or derivatives of the information is not required. Each computer has free access as the need arises.

WHERE IS SOFTWARE GOING?

The focus on the future will continue to be related to finding smaller and smaller computers and peripherals, having greater and expanded capabilities, for use on the workdesk or in the pocket. Where do software languages fit within this scenario? Two lines of questions should be asked when trying to determine the future of software languages. First, has the definition "computer language" changed from its initial interpretation? If it has changed, what is its new meaning? Will the new meaning change before 2010? And second, has software been able to erase the stigma ingrained by early computer developers? Will software be the driving force in 2010?

1. The Random House College Dictionary defines language as any set or system of formalized symbols, signs, gestures, or the like, used or conceived as a means of communications. The first part of the definition, formalized symbols (or texts), bears a great resemblance to the software developer's idea of first, second and third generation languages. Text information in the form of instructions and/or commands was provided to manipulate the hardware to perform communications between the users and the computer. In the later stages of the third phase, "speech" was included as a means to communicate between the two. 4GL HOLs and 4GL tools will have improved the manner in which the communications are performed, but only to the extent of allowing instructions to emulate more and more machine language code. From here a line is drawn. 5GLs and expert systems constructs will be the departure point from the equation, language equals instructions. In the year 2010, two other areas will take the industry further away from the present day definition of languages - signs (symbolic image processing) and gestures or the like (thought). Both areas of research have limited following; neither will be marketable products by that time. However, both will be the basis for the "languages" of the future.

2. From all the improvements, changes, enlightenments, and directions hardware has provided the computer industry, the fact remains hardware is only two dimensional. Computers and their peripherals continue to get smaller, with greater capability. Chip technology, Direct Access Storage Devices, switches, all have improved dramatically. But the bottom line still remains the same. Hardware is seen as trying to break the speed and physical barriers. When the long term arrives, when a thinking machine finally appears, it will not be based on hardware advancements, but on software.

BIBLIOGRAPHY

CHANG, Shi-Kuo, "Visual languages: a tutorial and survey", IEEE, January 1987, pp 29-39.

COLMERAUER, Alain, "Prolog in 10 figures", Communications of the ACM, December 1985, pp 1296-1310.

CUMMINGS, Steve, "Micro languages help businesses customize software", PC Week, August 1988, pp 100(2).

CUMMINGS, Steve, "Alternate languages address special needs", PC Week, January 1987, pp 70(2).

DeVRIES, F.J., "A functional program for the Fast Fourier transform", SIGPLAN, January 1988, vol 23, no 1 pp 67-74.

DRESSLER, Fritz R.S., "Knowledge crunchers - the fifth generation of computing", The Seybold Outlook on Professional Computing, June 1986, pp 15(4).

FALK, Howard, "Revisions and additions thrust Ada into real time", Computer Design, November 1988, pp 26(3).

FINN, V.K., "Logical information-processing systems", Nauchno-Tekn. Inf. Ser., 1986, vol 20, no 1 pp 7-10.

FONTANA, Maxine, et al, "Elements of expert system shells", PC Tech Journal, May 1988, pp 63(2).

GEAR, C. William, "Computer Organization and Programming", McGraw-Hill Book Company, 1969.

HALBERT, Daniel C., et al, "Using types and inheritance in object-oriented programming", IEEE Software, September 1987, pp 71(9).

HINDIN, Harvey J., "Designers choose from software-development environments", Computer Designs, September 1988, pp 74(4).

HUYNH, Tien, "An execution architecture for FP", IBM Journal of Research and Development, November 1986, pp 609(8).

JONES, Capers, "A new look at languages", Computerworld, November 1988, pp 97 (4).

KING, K.N., "Modula-2 programming", Georgia Institute of Technology, 1984.

LEWIS, Robert, "A Prolog to the future", PC World, December 1986, pp 284(12).

MCDONALD, C.S., "fsh-a functional UNIX command interpreter", Software Practice Exp. (UK), October 1987, vol 17, no 10, pp 685-700.

POWELL, Micheal, "Still cracking the whip", Computerweekly, May 1987, pp 26(2).

RUBINOFF, R., "Adapting MUMBLE: experience with natural language generation", (USSR), 1987, vol 2, pp 1063-1068.

SMOLKA, Scott, et al, "A CCS semantics for NIL", IBM Journal of Research and Development, September 1987, vol 31, pp 556(15).

SWEET, Frank, "Database directions", Computerworld, November 1988, pp 85(5).

URBAN, Frank E., "Computer languages", Handbook of Software Engineering, 1984, pp 184-200.